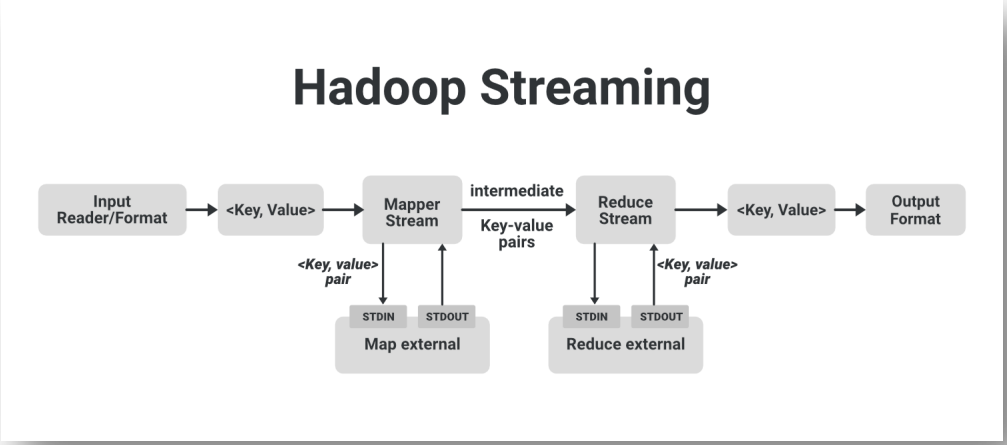
IES de Teis
Avda. de Galicia, 101
36216 – Vigo

Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

WordCount en Python

Hadoop, y en concreto la capa de procesamiento distribuido YARN, proporciona soporte para todos los lenguajes de programación que puedan leer y escribir en las entradas y salidas estándar.

La ejecución se realizará mediante la "Hadoop Streaming API" que nos ayudará a que los datos fluyan entre los mapas y los reducers vía STDIN y STDOUT. Para ello solamente será necesario usar en nuestro código Python "sys.stdin" o "sys.stdout". Hadoop Streaming se hará cargo del resto.



Para lanzar un MapReduce con Hadoop Streaming API será necesario invocar al archivo "hadoop-streaming.jar" y pasarle 4 parámetros obligatorios:

- -input y la ruta de la carpeta o archivo desde donde cargará datos el mapper
- -output la ruta de la carpeta donde los reducers escribirán sus datos de salida.
- -mapper y la ruta del ejecutable que contiene el código del mapper.
- -reducer y la ruta del ejecutable que contiene el código del reducer.

Opcionalmente podemos añadir más parámetros:

- -file y la ruta del archivo que queremos hacer disponible localmente en lo nodos de cómputo.
- -numReduceTasks especifica el número de reducers.

Para lanzar un MapReduce tendríamos que ejecutar el siguiente comando

IES de Teis
Avda. de Galicia, 101
36216 – Vigo

Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es


```
yarn jar /home/hadoop/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
-input /trabajo01/entrada/ \
-mapper mapper.py \
-file mapper.py \
-output /salida \
-reducer reducer.py \
-file reducer.py \
```

WordCount en Python.

En nuestro contador de palabras Python separaremos en archivos distintos el código relacionado con los mappers y el código relacionado con los reducers.

El código para nuestro mapper Python será el siguiente:

```
#!/usr/bin/python3

import sys

for line in sys.stdin:
  line = line.strip()
  words = line.split()
  for word in words:
    print (word, "\t", 1)
```

Para probar el código Podemos probarlo en una terminal ejecutándolo directamente con el siguiente comando.

```
python3 mapper.py
```

El resultado de este comando es que el equipo se quedará esperando una entrada por teclado, el stdin. Podemos introducir el texto que queramos hasta que pulsemos Intro, después esa entrada se procesará en el mapper, nos devolverá los resultados de trocear la línea en palabras acompañadas de un 1 y quedará listo para la siguiente línea. Lo podemos para en cualquier momento con la combinación de teclas Ctrl+C.

```
administrador@nodo01:~/trabajo01$ python3 mapper.py
Hola, esto es una prueba
Hola,    1
esto     1
es       1
una      1
prueba   1
aquí aparece otra línea
aquí     1
aparece          1
otra     1
línea    1
^CTraceback (most recent call last):
  File "/home/administrador/trabajo01/mapper.py", line 5, in <module>
    for line in sys.stdin:
KeyboardInterrupt

administrador@nodo01:~/trabajo01$
```

IES de Teis
Avda. de Galicia, 101
36216 – Vigo

Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

Otra manera de probar el código es pasándole los datos de entrada a través de una tubería.

```
cat datos.txt | python mapper.py
```

Técnicamente este comando lista el contenido del archivo “datos.txt” que debería salir por la pantalla, la salida estándar (stdout) pero como hemos colocado una tubería, esta salida se convierte en entrada estándar (stdin) del siguiente comando, la ejecución del código Python.

El código Python del reducer sería:

```
#!/usr/bin/env python

import sys

contador = 0
palabra_anterior = None

for linea in sys.stdin:
   linea = linea.strip()
   lista = linea.split("\t")
   palabra = lista[0]
   #apariciones = lista[1]
   #palabra, apariciones = linea.split("\t")

   if palabra_anterior == None:
      palabra_anterior = palabra


   if palabra == palabra_anterior:
      contador += 1

   else:
      print(palabra_anterior, "\t", contador)
      palabra_anterior = palabra
      contador = 1

print(palabra_anterior, "\t", contador)
```

IES de Teis
Avda. de Galicia, 101
36216 – Vigo

Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

Probar el reducer es algo más complicado porque necesitamos o bien generar un archivo con todos los casos de prueba o bien encadenar una secuencia de tuberías que una toda la secuencia de etapas mapper, shuffle y reduce.

Si optamos por usar una tubería debemos encargarnos de la ordenación de la etapa shuffle usando el comando sort de Linux. El comando de prueba sería el siguiente:

```
cat datos.txt | python3 mapper.py | sort | python3 reducer.py
```

```
administrador@nodo01:~/trabajo01$ cat datos.txt | python3 mapper.py | sort | python3 reducer.py
cuatro          1
dos     3
tres    2
uno  4
administrador@nodo01:~/trabajo01$
```

Si queremos evitar tener que llamar al intérprete de Python podemos dar a ambos archivos permisos de ejecución con el siguiente comando:

```
sudo chmod +x mapper.py
sudo chmod +x reducer.py
```

```
administrador@nodo01:~/trabajo01$ cat datos.txt | ./mapper.py | sort | ./reducer.py
cuatro          1
dos     3
tres    2
uno  4
administrador@nodo01:~/trabajo01$
```

Si nos fijamos en estas capturas de resultado de los reducers y en la captura anterior del mapper veremos que el resultado es correcto porque ha agrupado y contado los iguales. El resultado es el esperado, un contador de palabras.

Ahora que ya lo tenemos probado en local vamos a lanzarlo dentro de hadoop para que procese de manera distribuida.

En primer lugar es necesario subir los datos de entrada al sistema HDFS mediante el siguiente comando

```
hdfs dfs -put datos.txt /
```

donde datos.txt hace referencia a la ruta local donde están los datos de entrada y / es la ruta HDFS donde queremos subirlo, en este caso en la raíz del sistema de ficheros HDFS.

Con esto ya podemos lanzar el comando para ejecutar nuestros mappers y reducers a través del streaming de hadoop.

```
yarn jar /home/administrador/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
-input /datos.txt \
```

```
-mapper mapper.py \
-file mapper.py \
-output /salida \
-reducer reducer.py \
-file reducer.py
```

El commando anterior tiene bastantes partes pero todas son necesarias y de sentido común.

En la primera línea tenemos el archivo jar que vamos a ejecutar. Como podemos ver en su nombre (hadoop-streaming) este archivo es el que nos permitirá sustituir los mappers y los reducers por los que implementemos en otro lenguaje de programación.

La siguiente línea, la que empieza por “input” contiene la ruta HDFS donde se encuentra los datos de entrada. Esta ruta puede apuntar a un solo archivo como en este caso o a un conjunto de ellos usando comodines como el asterisco (*).

A continuación tenemos la línea que empieza por “mapper” en la que indicamos la ruta local donde está el archivo Python o de cualquier otro lenguaje de programación que realizará la función de mapper.

La siguiente línea empieza por “file” y su misión es anexar el archivo indicado al contender del nodo donde se va a ejecutar ese mapper. Recordemos que la idea del procesamiento distribuido es que las instrucciones vayan a los datos. Por lo tanto lo que hace esta línea es incluir el mapper en los contenedores para que cualquier nodo tenga disponible y pueda ejecutar ese preciso mapper.

Seguidamente tenemos la línea “output” donde especificamos la ruta HDFS donde se creará la carpeta con los resultados de los reducers. Aquí estará la solución.

La siguiente línea es la de “reducer” en la que se indica el nombre del archivo Python o cualquier otro lenguaje de programación que realizará las labores de reducer.

Por último repetimos nuevamente el parámetro “file” para indicar que también queremos anexar el archivo con el código del reducer para que se pueda ejecutar en cualquiera de los nodos.

Los parámetros “file” no serán necesarios si los mappers y reducers ya están guardados en todos los nodos.